CUDA Parallel Algorithms for Forward and Inverse Structural Gravity Problems

Alexander Tsidaev

Bulashevich Institute of Geophysics, Yekaterinburg, Russia
pdc@tsidaev.ru

Abstract. This paper describes usage of CUDA parallelization scheme for forward and inverse gravity problems for structural boundaries. Forward problem is calculated using the finite elements approach. This means that the whole calculation volume is split into parallelepipeds and then the gravity effect of each is calculated using known formula. Inverse problem solution is found using iteration local corrections method. This method requires only forward problem calculation on each iteration and does not use the operator inversion. Obtained results show that even cheap consumer video cards are highly effective for algorithm parallelization.

Keywords: geophysics · parallel algorithm · forward gravity problem · inverse gravity problem · CUDA

1 Introduction

Gravity field measurement and interpretation is one of the simplest geophysical methods. It does not require additional preparations (unlike, e.g., the method of artifical magnetization in magnetometry) or large amount of sensors (unlike seismometry). But despite this apparent simplicity, the interpretation of big amounts of data is very hard. For a long time only 2D density models along profiles were constructed, but now we have a need in 3D models construction as they are much more informative.

3D geophysical problems solution usually requires high-performance calculations, especially if input data is specified on a big grid. In many cases the performance of regular personal computers is not enough, this leads to the need of supercomputers usage. But currently the supercomputers are not widely distributed and accessing one of them can be problematic. Even taking into account that the data can be sent over the Internet to any distant supercomputer, the big amount of data to send makes the process very slow.

But now we can use a different parallelization approach. With big popularity of videogames, a standard video card of personal computer became much more than just a display adapter. It is a mini-computer with higher performance than central processing unit (CPU) itself. Performance of top-class graphical processing units (GPU) is similar to the performance of 10-years old supercomputers.

But even cheap consumer GPU can provide a high speed up of calculations if program was parallelized correctly.

Compute Unified Device Architecture (CUDA) is a parallelization technology by one of the leading GPU producers, nVidia. Current paper is dedicated to the usage of CUDA for the creation of efficient programs for gravity geophysical problems. Section 2 contains description of gravity structural forward problem and algorithm for its parallelization using CUDA. Section 3 describes the inverse gravity problem for structural boundaries and the idea of local corrections method. In the 4 section the comparison of different calculation schemes is provided.

2 Forward gravity problem for structural boundary

In general, gravity potential that is produced by some object is calculated as (for details one may refer to, e.g., [1])

$$W(x,y,z) = \gamma \int\limits_V \frac{\sigma(\xi,\eta,\zeta)dV}{\sqrt{(\xi-x)^2+(\eta-y)^2+(\zeta-z)^2}} \tag{1}$$

Here x, y, z - point of observation; $\sigma(\xi,\eta,\zeta)$ is the density value at a point (ξ,η,ζ) under Earth surface; $\gamma = 6.67408 \cdot 10^{-11} m^2 \cdot kg^{-1} \cdot s^{-2}$ is a gravitational constant. z axis direction is downwards.

If we neglect the sphericity of Earth, the first derivative by z of (1) can be taken as anomaly gravity field Δg on Earth surface. This is the main formula for gravity forward problem:

$$\Delta g = W'_z(x,y,z) = \gamma \int\limits_V \frac{\sigma(\xi,\eta,\zeta)(\zeta-z)dV}{[(\xi-x)^2+(\eta-y)^2+(\zeta-z)^2]^{3/2}} \tag{2}$$

It is clearly seen from (2) that gravity field on Earth level depends only on density distribution. While density values may vary between neighbour points, in regional studies of big squares we can ignore this variance. In such scales Earth gradiental density distribution could be replaced with the structural model of homogeneous layers with constant densities (Fig. 1(a)). In [2] it is shown that the topography of structural boundaries between these layers is the only element, which produces non-constant gravity field in this model (Fig. 1(b)). This is 3D analogue of Strakhov boundaries parameterization, which was presented in [3].

As the result, we do not need to know absolute density values for layers and the gravity effect of structural boundary could be calculated up to a constant as

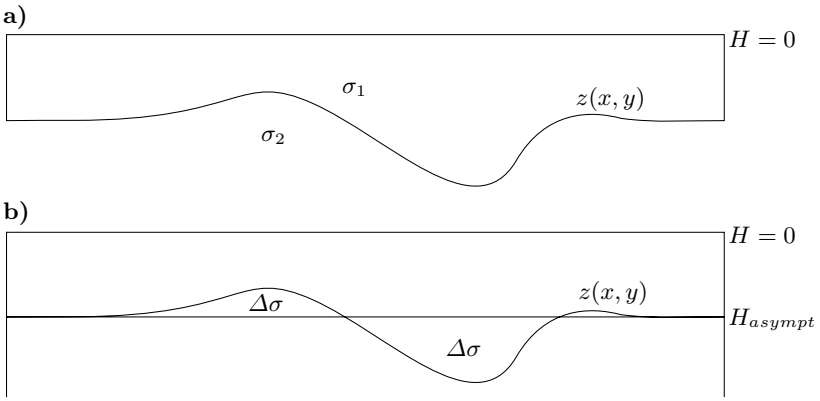

Fig. 1. 2-layer model with absolute density values (a) and its parameterization (b)

$$\Delta g = U(x,y,z) = \gamma\Delta\sigma \int\limits_{-\infty}^{\infty}\int\limits_{-\infty}^{\infty} \frac{dxdy}{\sqrt{(x-x')^2+(y-y')^2+(z-z')^2}}\Bigg|_{z(x,y)}^{H} =$$
$$= \gamma\Delta\sigma \int\limits_{-\infty}^{\infty}\int\limits_{-\infty}^{\infty} \Bigg(\frac{1}{\sqrt{(x-x')^2+(y-y')^2+z(x,y)^2}} - \tag{3}$$
$$- \frac{1}{\sqrt{(x-x')^2+(y-y')^2+H^2}}\Bigg)dxdy$$

here (x', y') - point of observation on Earth level (where $z' = 0$); $z(x, y)$ - structural boundary surface; H - asymptote of $z(x, y)$, $\Delta\sigma$ - density jump on a surface $z(x, y)$, it is equal to the difference $(\sigma_1 - \sigma_2)$ between density values of layers above (σ_1) and below (σ_2) the structural boundary.

Direct calculation using formula (3) is complicated by the weak singularity in the point $(x = x', y = y', z(x, y) \cup H = 0)$, so the formula is inapplicable for the shallow surfaces. To avoid this limitation in [2] we offered an idea of more stable and efficient structural gravity problem calculation using the finite elements approach. We split the observation plane into a set of rectangular elements $x_i = \xi_i, y_i = \eta_i$ $(0 \le i < N, 0 \le j < M$, the same discretization is used for the boundary too) and calculate a field $\Delta g(x', y')$ as a sum

$$\Delta g(x', y') = \gamma\Delta\sigma \cdot \sum_{ij} \Delta g_{ij}(x', y') \tag{4}$$

of gravity effects of parallelepiped elements (whose formula could be obtained analytically)

$$\Delta g_{ij}(x', y') = (\eta - y') \cdot ln(|\xi - x'| + R) + (\xi - x') \cdot ln(|\eta - y'| + R) - \\ - \zeta \cdot arctg \left[\frac{(\xi - x') \cdot (\eta - y')}{\zeta \cdot R} \right] \Bigg|_{\xi_i}^{\xi_{i+1}} \Bigg|_{\eta_j}^{\eta_{j+1}} \Bigg|_{H}^{z(\xi_i, \eta_j)} \tag{5}$$

Here $R = \sqrt{(\xi - x)^2 + (\eta - y)^2 + \zeta^2}$.

In addition to the possibility to calculate field of near-surface boundaries, this method has another advantage over direct calculation using (3): there is no need for the "real" asymptote for a boundary. Since the boundary gravity effect calculation is actually replaced with calculation of gravity effect for the limited body, any horizontally located plane could be used instead of asymptote H. But it is better to take the average value of the boundary: for all other H the field will contain additional constant component, which is equal to the gravity effect of a flat layer located between H and boundary average depth.

It is obvious that the algorithmic implementation of (4) should contain two nested loops: one iterates through points of observation (x', y'), and another iterates through the surface discretes (ξ, η). This is very close to the CUDA internal structure, which has blocks and threads [4]. Programmer can run any number of calculation blocks, and each block will execute the same program code in parallel in predefined (and permanent for all blocks) number of threads. So if we run calculation of (5) on $N \cdot M$ blocks in $N \cdot M$ threads and then summarize results for all threads inside each block, the combination of the block results will be exactly the gravity effect of the structural boundary (up to $\gamma \Delta\sigma$). The described scheme is presented on Fig. 2.

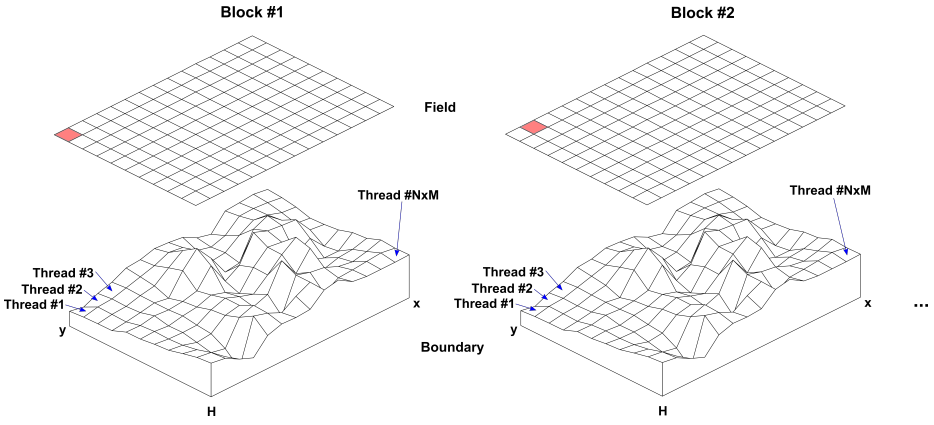

Fig. 2. Blocks and threads parallelization scheme: each block calculates observed field value in specific point (*top*), each block thread calculates the gravity effect of single parallelepiped (*bottom*).

This method also provides a good scalability across multiple GPUs. Each graphic card can calculate gravity field in a separate set of grid rows, and host then can easily combine results from different GPUs together. Number of graphic cards in single PC is limited by the number of physical PCIe slots and PCIe channels, but it is almost always possible to insert at least two GPUs in one motherboard. Another option is to use several CUDA-enabled PCs connected with MPI (Message Passing Interface) over a local network. But one should control the performance of the process due to a big overhead of network operations.

3 Inverse gravity problem for structural boundary

Inverse gravity problem occurs when it is needed to restore density distribution by known (observed) gravity field. This problem has big practical necessity because it allows one to explore Earth structure mathematically, without such costly operations as borehole drilling. But the problem is ill-posed so the solution is not unique (there are continuum of solutions that produce the same gravity effect) and the solution process of the problem is unstable.

Direct inversion of (3) gives us the Fredholm equation of first kind. It can be reduced to a system of linear equations, which then could be solved using Tikhonov regularization scheme and approximate methods. This approach has a good description in [5], one of possible parallelization schemes is provided there as well. But for the big grids this system of linear equations will be very large (i.e. $N^2 \cdot M^2$) and so some additional algorithms would be needed to operate such a big data. This is the reason to use iterative method of local corrections.

The method of local corrections [6, 7] was developed by I. Prutkin and its idea is that the main contribution to the gravity field at some point (x, y) is introduced by the masses located directly under that point. Since the boundaries between density layers rarely have large gradients, this assumption can be considered as true. Main formula of the method is simple:

$$z_{xy}^{n+1} = \frac{z_{xy}^{n}}{1 + \left(\alpha / \gamma c \Delta\sigma\right) \cdot z_{xy}^{n} \left(U_{xy} - U_{xy}^{n}\right)} \tag{6}$$

here z_{xy}^{n} is the position of a boundary z in a point (x, y) on nth iteration; U_{xy} is the observed field in the point (x, y); U_{xy}^{n} is the gravity field of nth iteration boundary z_{xy}^{n}; c is cubature formula coefficient; α is regularization parameter.

Parameter α is very important. This is not the regularization parameter in Tikhonov's sense. It acts more like stabilizing factor, which does not allow boundary to deviate too far from the previous approximation. It should be selected relatively small, $0 < \alpha \leq 1$.

I. Prutkin selected the estimated asymptote level for a $z(x, y)$ as the initial approximation z^0. In [2] we offered modified technique. We construct initial approximation using a priori data obtained using another geophysical methods (e.g. seismic surveys). This reduces the number of iterations and also forces the method to produce more geologically meaningful solutions.

It is clearly seen from (6) that the only calculation that is needed on each iteration of local corrections method is the forward gravity problem solution for a density boundary of previous approximation. So the parallelization idea is quite simple and relies on parallelization algorithm described in section 2:

1. Calculate field of the boundary of nth approximation using CUDA parallel scheme.
2. Calculate difference $U - U^n$ between observed and obtained fields.
3. If mean value of the difference is less than some preselected ε, then exit. Current approximation of $z(x, y)$ is the solution.
4. Else calculate $(n + 1)$th approximation of the boundary by (6) and repeat algorithm from the first step.

4 Calculations and results

Algorithms were implemented for different hardware to evaluate efficiency. Program for CPU was written in C++ and compiled with gcc (g++ -o3) compiler. GPU implementation is also in C++ and CUDA part was compiled with proprietary nvcc compiler while gcc has been used for the host code. The grid with of 256x256 nodes was taken as input data. Tesla M2050 GPUs were accessed on "Uran" supercomputer, which is located in Krasovskii Institute of Mathematics and Mechanics (Yekaterinburg). Results of testing are presented in Table 1.

Table 1. Duration of forward problem calculation on different hardware

Hardware	**Calculation time, s**	**Speed up factor**
1 CPU core (Intel Xeon E5520)	3968.16 (66+ minutes)	1x
1 GPU (nVidia GTX580)	62.412	64x
1 GPU (nVidia GTX780 Ti)	32.711	121x
1 GPU (Tesla M2050)	27.841	143x
2 GPUs (nVidia GTX780 Ti)	16.862	235x
2 GPUs (Tesla M2050)	13.693	290x
4 GPUs (Tesla M2050)	7.528	527x
8 GPUs (Tesla M2050)	5.274	752x

So for a big grids calculations on CPU are almost unacceptable. Though CPU still could be used for a single forward problem calculation, its usage in iterative process of inverse problem solution entails long waiting time. Local corrections method requires α parameter preselection which could be performed only empirically, so the number of required iterations (usually 20-30) is multiplied by the amount of experiments. This makes CPU usage for inverse problem calculation highly problematic.

Conversely, even relatively cheap outdated consumer GPU nVidia GTX580 has acceptable time of calculation. And modern consumer devices like nVidia GTX780 Ti can compete even with previous generation of specialized CUDA

calculators (Tesla). The one who has no access to supercomputer can easily build high-performance computer himself using these widely available devices.

As it is seen from Table 1, the dependence between number of GPUs and calculation time is almost linear for 1, 2 and 4 Tesla cards. Regularity break on 8 GPUs is related to the increased overhead of host operations. For a bigger grids the linear nature of dependence returns. For example, for a grid of size 1335x1404 the calculation times on 4 and 8 GPUs are 20m12s and 9m47s respectively. This linear dependence confirms very good scalability of CUDA calculations: with twofold GPU count increase the duration of calculation is reduced in 2 times too.

5 Conclusion

Gravity modeling is very important geophysical task. In this paper the algorithms for forward and inverse problems parallelization were offered. Obtained results show that GPU calculations are very effective way to lower the calculation time. Even widespread graphic cards, which initially were intended for a videogame use, can be utilized for a high-performance calculations. Simpleness of CUDA make this technology available for a mass use while scalability allows one to increase performance easily.

The research was supported by the Russian Science Foundation (project 14-27-00059).

References

1. Blokh, Yu.I.: Interpretation of gravity and magnetic anomalies (In Russian), `http://sigma3d.com/pdf/books/blokh-interp.pdf` (2009)
2. Martyshko, P.S., Ladovskiy, I.V., Tsidaev, A.G.: Constructional of Regional Geophysical Models Based on the Joint Interpretation of Gravity and Seismic Data. Izvestiya, Physics of the Solid Earth, 2010, Vol. 46, No. 11, 931–942 (2010)
3. Strakhov, V.N.: On the parameterization problem in gravity inversion (In Russian). Fizika Zemli, No. 6, 39–50 (1978)
4. Wilt, N.: CUDA handbook: A comprehensive guide to GPU programming. Addison-Wesley Professional, Boston (2013)
5. Akimova, E.N., Vasin, V.V., Misilov, V.E.: Algorithms for solving inverse gravimetry problems of finding the interface between media on multiprocessing computer systems (In Russian). Vestnik UGATU, vol. 18, no. 2 (63), 208–217 (2014)
6. Prutkin, I.L.: The solution of three-dimensional inverse gravimetric problem in the class of contact surfaces by the method of local corrections. Izvestiya. Phys. Solid Earth 22 (1), 49-55 (1986)
7. Prutkin, I., Saleh, A.: Gravity and magnetic data inversion for 3D topography of the Moho discontinuity in the northern Red Sea area, Egypt. Journal of Geodynamics 47 (5), 237-245 (2009)